

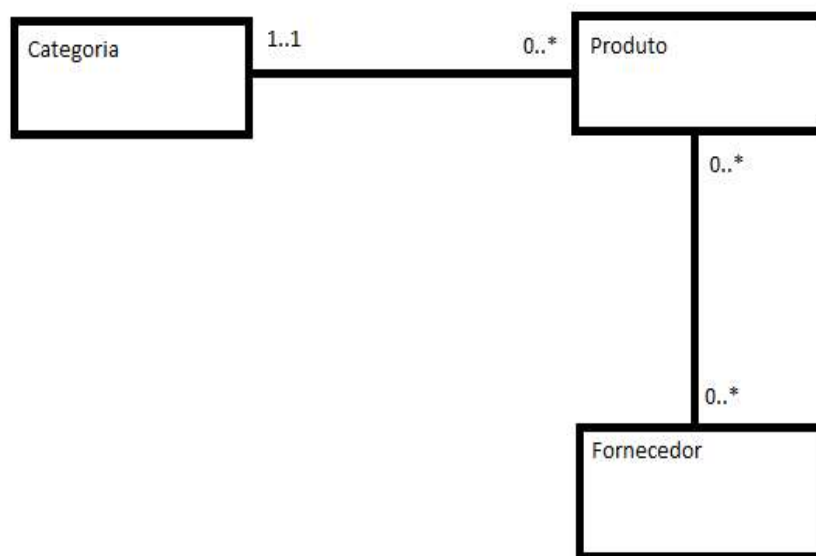
Fatec Praia Grande

Paulo R. T. Cândido

NodeJS - parte 3

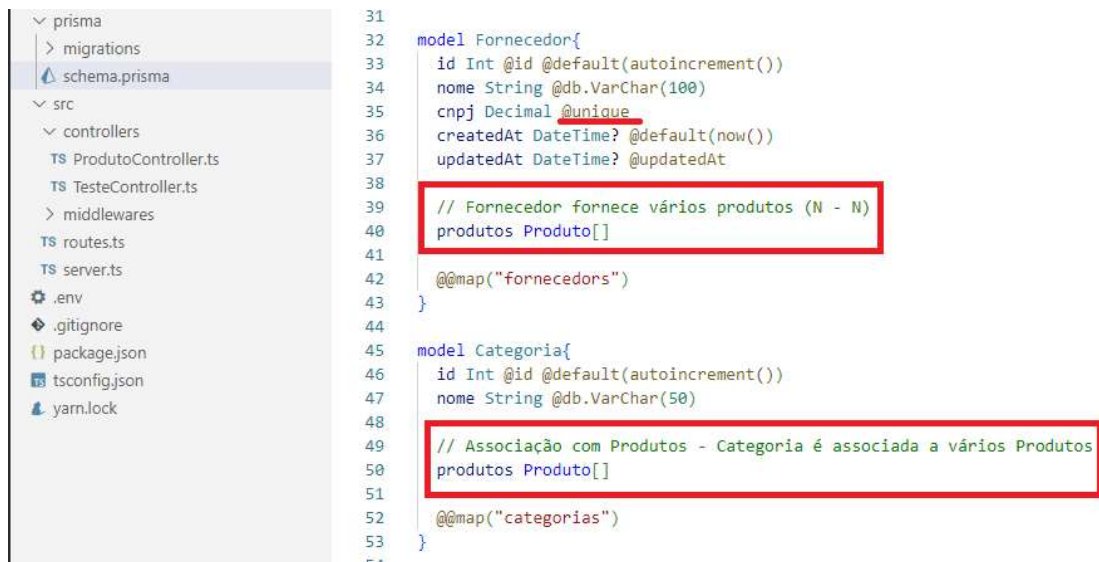
Models associados

1) Definir os models



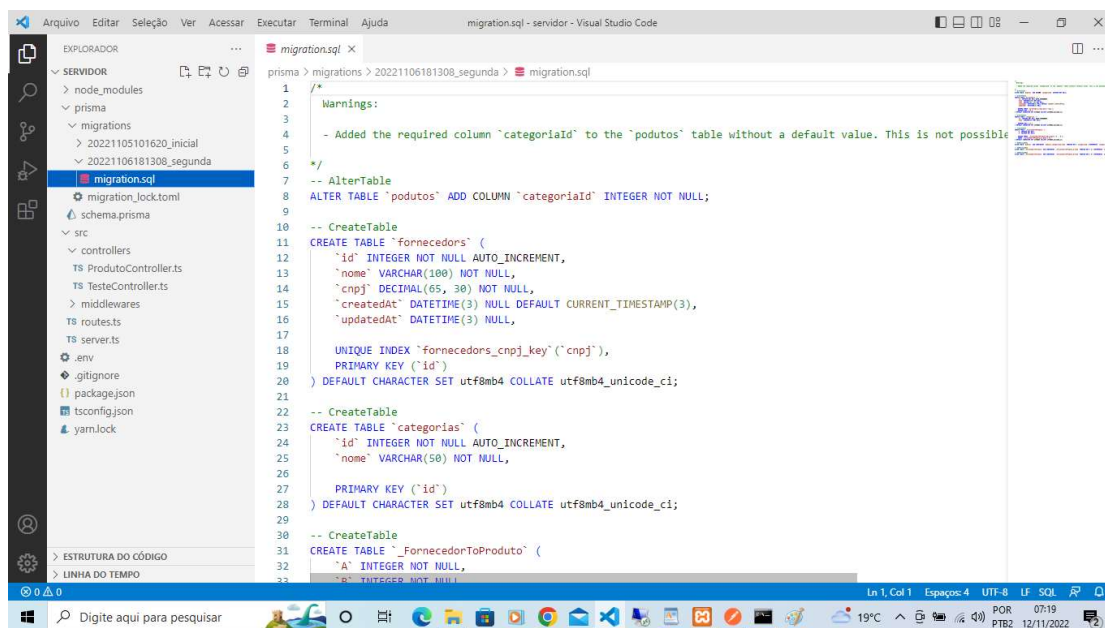
The screenshot shows a code editor with a file explorer on the left and code on the right. The file explorer shows a project structure with files like migrations, schema.prisma, src, controllers, routes, server, .env, .gitignore, package.json, tsconfig.json, and yarn.lock. The code on the right is a Prisma client configuration and schema definition. A red box highlights the schema definition for the Produto model, showing its fields and relationships.

```
3
4 generator client {
5   provider = "prisma-client-js"
6 }
7
8 datasource db {
9   provider = "mysql"
10  url       = env("DATABASE_URL")
11 }
12
13 // Para mais detalhes: https://www.prisma.io/docs/concepts/components/prisma-schema
14
15 model Produto {
16   id Int @id @default(autoincrement())
17   nome String @db.VarChar(100)
18   preco Decimal @db.Decimal(12,2) // independe do fornecedor, talvez não seja o mais correto, mas é só um exemplo
19   createdAt DateTime? @default(now())
20   updatedAt DateTime? @updatedAt
21
22   // Associação com Categoria - Produto é de uma única Categoria (N -> 1)
23   categoriaId Int
24   categoria Categoria @relation(fields: [categoriaId], references: [id])
25
26   // Produtos tem vários fornecedores (N - N)
27   fornecedores Fornecedor[]
28
29   @@map("produtos")
30 }
```



2) Executar migrações

yarn prisma migrate dev --name segunda



3) Alterar controller de produtos

Função index

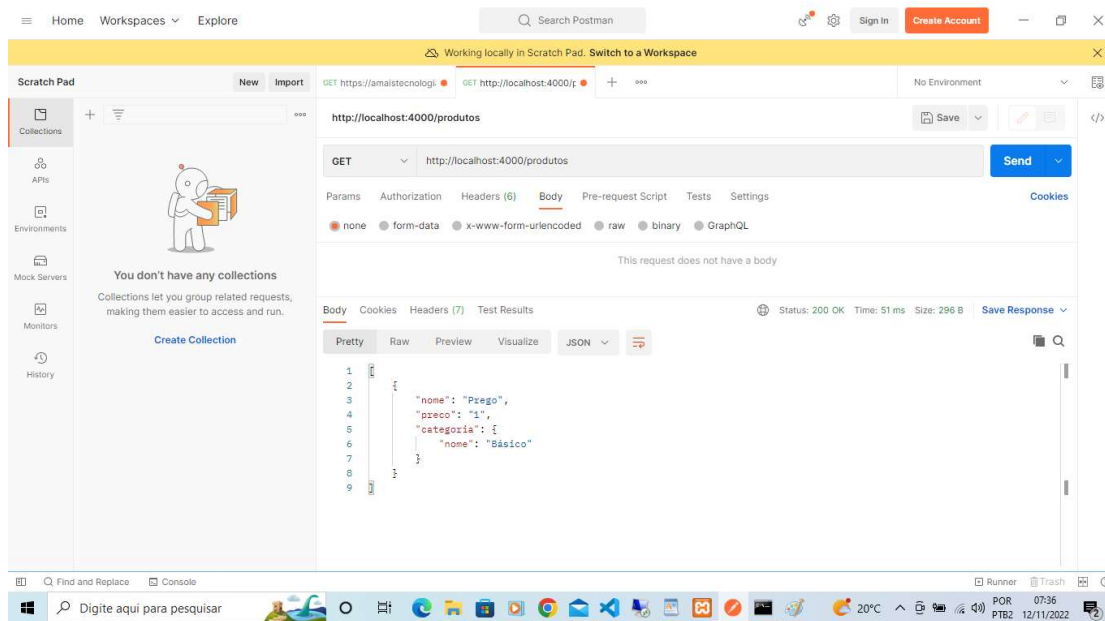
```
async index(req:Request,res:Response){
  const prisma = new PrismaClient();
  const produtos = await prisma.produto.findMany( // recupera todos os produto
  {
    orderBy:{nome:'asc'},
    select:{
      nome:true, // seleciona as propriedade desejadas de Produto
      preco:true,
      categoria:{
        select:{nome:true} // traz do model relacionado Categoria apenas o nome
      }
    }
  })
}
```

```

    }
  }
};
res.status(200).json(produtos);
}

```

Obs: para o teste abaixo foi cadastrado ditetamente no banco de dados categoria com id=1 e nome="Básico"

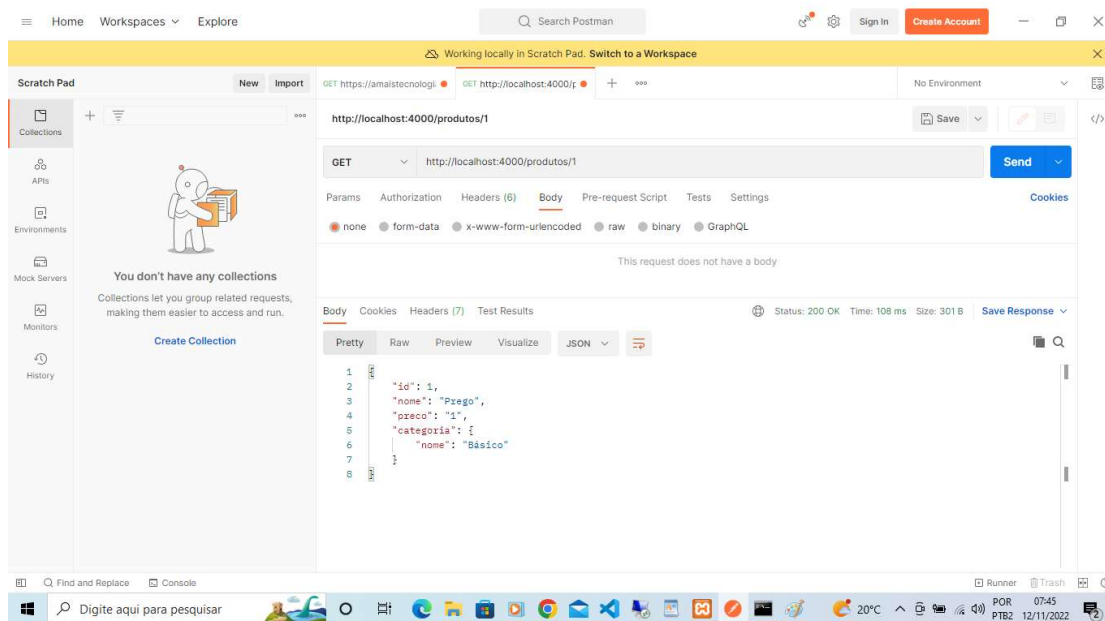


Função show

```

async show(req:Request,res:Response){
  const prisma = new PrismaClient();
  const produto = await prisma.produto.findUnique( // busca produto conforme where
    {
      where:{id: Number(req.params.id)},
      select:{
        id: true, // seleciona as propriedade desejadas de Produto
        nome:true,
        preco:true,
        categoria:{
          select:{nome:true} // traz do model relacionado Categoria apenas o nome
        }
      }
    }
  );
  res.status(200).json(produto);
}

```

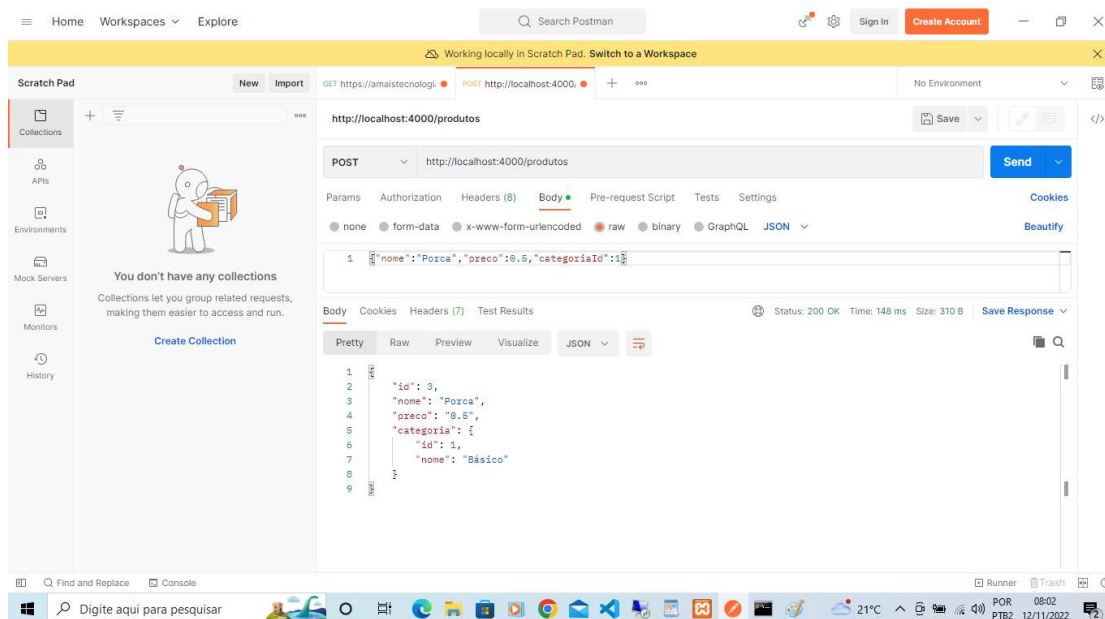


Função Store

```

async store(req:Request,res:Response){
  const prisma = new PrismaClient();
  //obtem json vindo do cliente. Exemplo Formato: {nome: "Prego", preco:2.3, categoriaId:1}
  const {nome, preco, categoriaId} = req.body;
  const novoPoduto = await prisma.produto.create(
    {
      data: {
        nome: nome,
        preco: preco,
        categoria: {connect:{id:categoriaId}} // associa produto à categoria
      },
      select: {
        id:true,
        nome:true,
        preco:true,
        categoria: true // traz todos os dados de categoria
      }
    }
  );
  res.status(200).json(novoPoduto);
}

```

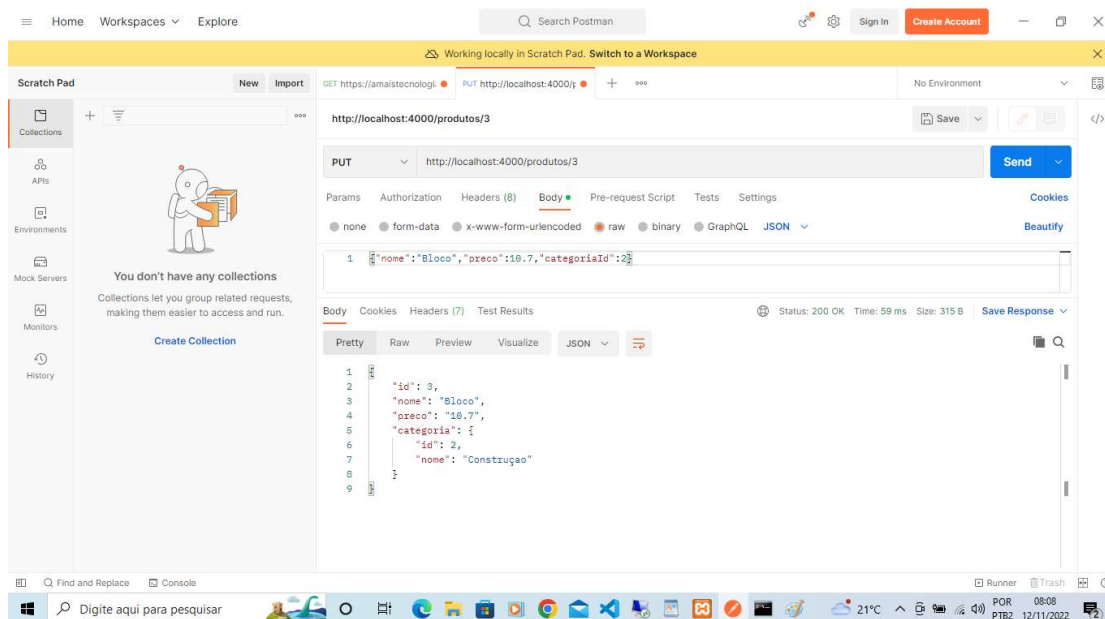


Função update

Obs.: para funcionar o teste abaixo é preciso inserir categoria id=2, nome="Construção"

```
async update(req:Request,res:Response){
  const prisma = new PrismaClient();
  const {nome, preco, categoriaId} = req.body;
  const produtoAlterado = await prisma.produto.update(
    {
      where: {id: Number(req.params.id) },
      data: {
        nome: nome,
        preco: preco,
        categoria: {connect:{id:categoriaId}} // associa produto à categoria
      },
      select: {
        id:true,
        nome:true,
        preco:true,
        categoria:true
      }
    }
  );
  res.status(200).json(produtoAlterado);
}
```

No exemplo é alterado o produto com id=3, informado na URL.



Função delete não precisa ser alterada.

4) Associar produtos a fornecedores

```

async associarFornecedores(req:Request,res:Response){
  // Exemplo de json recebido: {fornecedores:[1,2]}
  const {fornecedores} = req.body;
  const dados = fornecedores.map( (x:any)=>{return {id:x}} ); // resulta: [{id:1},{id:2}]
  const prisma = new PrismaClient();
  const produtoAlterado = await prisma.produto.update(
    {
      where: {id: Number(req.params.id) },
      data: {
        fornecedores: {connect:dados} // associa produto à categoria
      },
      select :{
        nome: true,
        preco: true,
        categoria: true,
        fornecedores: true
      }
    }
  );
  return res.status(200).json(produtoAlterado);
}

```

Cria rota

```

  controllers
  TS ProdutoController.ts
  TS TesteController.ts
  > middlewares
  routes.ts
  TS server.ts
  .env
  .gitignore
  package.json
  tsconfig.json
  yarn.lock

/ import ValidaTeste1 from './middlewares/ValidaTeste1';
8 // Instancia roteador
9 const Roteador = Router();
10 // Define rota tipo get que, para funcionar, deve ser requisitada conforme exemplo.
11 // Exemplo de requisição: localhost:4000/teste/123?num=456
12 // Onde 123 e 456 podem ser substituídos por quaisquer valores
13 Roteador.get(
14   // URL com parâmetro :id
15   '/teste/:id',
16   ValidaTeste1,
17   // Aciona função do TesteController
18   new TesteController().teste1
19 );
20
21 Roteador.get('/produtos', new ProdutoController().index);
22 Roteador.get('/produtos/:id', new ProdutoController().show);
23 Roteador.post('/produtos', new ProdutoController().store);
24 Roteador.put('/produtos/:id', new ProdutoController().update);
25 Roteador.delete('/produtos/:id', new ProdutoController().delete);
26 Roteador.put('/produtos/fornecedores/:id', new ProdutoController().associarFornecedores);
27
28 export default Roteador;
```

Obs.: para este teste, cadastrar 2 fornecedores diretamente no banco de dados

